CommWorks
a 3Com company

High Level and Detailed Design Specification

Wireless 3G NextGen (TC2K) Rel 1.0

PPS ID: 1186 Feature ID: FAS Sub-Agent

Feature Activation

FAS Sub Agent

APPROVALS

TITLE	Printed Name:	SIGNATURE:	DATE:
Product Management	Lee Rosenbaum		
Engineering Lead	Ching Y. Kung		
Program Manager	Carolyn Heide		
Architect	Christian Rigg		

Exact signatures required will be determined by the EMT Team

DOCUMENT CONTROL:_____RELEASE DATE:_____

RECORD OF
REVISIONS

ORIGINATOR:	REVISED SECTION/PARAGRAPH:	REVISION LEVEL:	RELEASE DATE:
Christian Rigg	Initial Version	0.3	XXXXXXX
Christian Rigg	Revised version after review	1.0	XXXXXXX

Note: This document is controlled electronically. Controlled hard copies are located in Document Distribution Points and/or bear a red "CONTROLLED" stamp. Other hard copies, unless specifically printed for an auditor's reference, are uncontrolled and invalid.

1 Purpose And Overview

This document describes the detailed design and implementation of the FAS Sub-Agent for Wireless 3G NextGen 1.0

1.1 Executive Summary

This section should list the summary of features supported.

2 Scope

2.1 Intended Audience

This Design Specification document is intended for technical managers, design engineers, test engineers, supporting engineers and technical publications in this project.

2.2 Environment/Infrastructure

There should be no substantial modifications to the Software Development environment, which was used to develop Wireless 2.0 and 2.1 releases. However, for a detailed understanding of the Software Development environment, readers are referred to the Software Development Plan (SDP) for this project.

2.3 Assumptions

This paragraph shall describe any assumptions that apply to this project that are not inherently obvious and which may include 3rd party software, partnerships, existing software and hardware modifications, and associated project timelines.

2.4 Limitations

There should be no substantial modifications to the Software Development environment, which was used to develop Wireless 2.0 and 2.1 releases.

3 Definitions

3.1 Acronyms and Abbreviations

This section explains and defines words and acronyms that are used throughout this document that are not found in the Glossary of Terms for Product/Technology Development.

Term	Definition
Activation	An action that allows a *Feature* to operate within a CommWorks system. *Activations* are performed u *Feature Key*.
Activation System	A service that informs the *Application* of *Feature States*.
Baseline	A partition separating common functionality and functionality assigned Feature IDs. I.e. features that feature key enabled are now part of the base code and permanently enabled.
BHCA	Busy Hour Call Attempts
Blade	A physical card in a chassis.
Capacity	A measure of potential capability that can be licensed for use.
CEM	Common Element Manager
COTS	Commercial Off The Shelf - in reference to equipment based on commercial HW such as Sun Servers NT based servers

CPS	CommWorks Professional Services
Deactivation	An action that disallows a *Feature* to operate within a CommWorks system. *Deactivation*s are perfor *Feature Keys*.
Dependence	A condition where one *Feature* depends upon another *Feature* to be functional.
Disable	Ability to turn off the feature via provisioning attributes. Dependent on feature being activated previ
EnableElement Feature Key	Feature Key that is node-locked to a particular Network Element
Enable	Ability to turn on the feature via provisioning attributes. Dependent on feature being activated previ
FAS	Feature Activation System
FAS Domain	The network domain that is managed by a FAS server
FAS Server	FAS component that manages Feature Keys on a customer network.
Feature	A group of related functions within a single *application* that implements a valuable service
Feature ID	An identifier unique within CommWorks for a *Feature,* across all lines of business.
Feature Key	A block of data that sets the *Feature State* of a single *Feature* associated with a unique *Feature ID.*
Feature Parameter	Information included in a *Feature Key* that stipulates specific behavior of the *Feature*, such as date, e date, feature ID, etc.
Feature State	The level of *Activation* that a *Feature* has. This should include any *Feature Parameters* associated w *Feature*. Refers to the settings of a *Feature* or the attributes that make up the functionality of the feat
FFD	Feature Functional Description (a document)
	Feature Key Generator
Java RMI	Java Remote Method Invocation
Network Feature Key	Feature Key that is not node-locked to a particular Network Element.
NMB	Network Management Bus. Intra-TC-1000 communications between the NMC card and the NAC ca serial communications star network based on UARTs (Universal Asynchronous Receiver Transmitter software download, and other network management messages are communicated over these serial lin
SCB	System Control Bus. Intra TC-2000 communications between Application Modules and the System M based on Ethernet.
SCOPS	Supply Chain Operations. A supply chain is a network of facilities and distribution that performs the procurement of materials, transformation of materials, and distribution of the finished products to cus Customer orders (including feature orders) go through the supply chain to get filled.
"shall"	"shall" in this document has the meaning of MUST. It indicates a high priority.
"should"	"should" in this document has the meaning of MAY. It indicates a medium priority.

SPI	Millennium Service Provider Interface
TC-1000	Total Control 1000
TC-2000	Total Control 2000
Trial Activation	A *Feature State* that is initially Active but becomes Inactive after a specified period without requiring intervention.

3.2 Definitions

Feature Activation System (FAS) – Feature Activation System consists of having a feature key to communicate what features are permitted to be activated and having processing entities that act on the activation request. The feature activation process can be envisioned as being handled by a combination of processing entities each of which has certain responsibilities. A part of the Feature Activation System must be present in each Network Element in order to allow the customer to apply activation without reliance on any system external to the Network Element.

Feature Key Generator - This is the component that generates Feature Keys. It is operated by CommWorks. A database is needed for storing Feature Key specific information. There is no direct communications between the Feature Key Generator and the customer's network.

Feature Identifier – This is the identifier for a particular feature category, like QOS, IPSEC, etc.

CEM - This is the SNMP manager that communicates to the SNMP Agents such as SM, FAS-Server, FAS-Agents, etc.

FAS Server - This is the component of the Network Management Station that distributes Feature Keys to the Network Elements. There is one FAS Server per customer network management domain (FAS Domain). It can reside on the same machine as the CEM Server. The FAS Server software is distributed together with the CEM software package.

FAS Agent – This is the subsystem in the Network Element that receives and stores Feature Keys. It divides and distributes Feature Keys to the Application Modules. It resides in the Network Element Manager (System Manager in Total Control 2000 system) part of the NE.

FAS Sub-agent – This is the subsystem in the Application Module that receives Feature Keys from the FAS Agent. In Total Control 2000 system, AM-agents shall have FAS-Subagent

Feature Key – This is a piece of data that is used by the FAS Server and the FAS Agent to activate a feature.

Feature Unit (FU) - One *Feature Unit* is the elementary unit (permission) of a feature. One Feature Unit is needed to activate one Feature Category on one FAS Sub-agent. The Feature Key includes an [1 to X] number of Feature Units (permissions).

Network Feature Key – Feature key(s) generated by the CommWorks Feature Key Generator in XML file format for the FAS Server. This is based on the serial number/host id of the machine where FAS Server is running. This is generated for customers.

Element Feature Key - Feature key(s) generated by the CommWorks Feature Key Generator or by FAS Server in XML file format. These are used by the FAS Agent (System Manager in Total Control 2000 system). This is based on the serial number of the control shelf in Total Control 2000 system. These keys are generated automatically by the FAS Server from the Network Feature Keys, or by CommWorks using the Feature Key Generator if the serial number of the control shelf is available.

System Manager (SM) - The System Manager provides the single point of network management for the Total Control 2000 system. It is responsible for storage and distribution of software loads and initial configuration for all the Total Control 2000 application blades. The System Manager also provides the SCB interconnects for supporting multi-shelf systems. SM has the FAS-Agent.

FAS Agent MIB – This is the feature activation system MIB on the FAS Agent. It has two tables –NE Feature Key Status Table and Blade Feature Key Status Table.

NE Feature Key Status Table - contains objects that are read-only. This table gives the overall picture of the total number of feature units available per feature units and their availability/allocation. This table is contained within FAS Agent MIB.

Blade Feature Key Status Table – contains objects that are read-only. This table gives the overall picture of feature allocation on each FAS sub-agent within FAS Agent control. This table provides information which blade has been activated which feature identifier, and how many feature units per feature identifier are in that blade. It also tells if the features are enabled in those blades.

FAS Sub-agent MIB – This is the feature activation system MIB for the FAS sub-agent. Each sub-agent blade owns this MIB. A module feature configuration table is in this MIB. This is a read-create MIB.

Feature Serial Number – This is a serial number embedded in the feature key that gets generated by the Feature Key Generator at CommWorks. The feature key serial number is maintained by CommWorks for each feature identifier issued. FAS-Server propagates the feature key serial number from Network Key to the Element Keys.

4 Applicable Documents and External Standards Specifications

This section shall list the number, title, revision and date of all documents referenced in this High Level / Detailed Software Design Specification. This section should include the list below as well as any other applicable documents (i.e. international standards, federal standards, etc.).

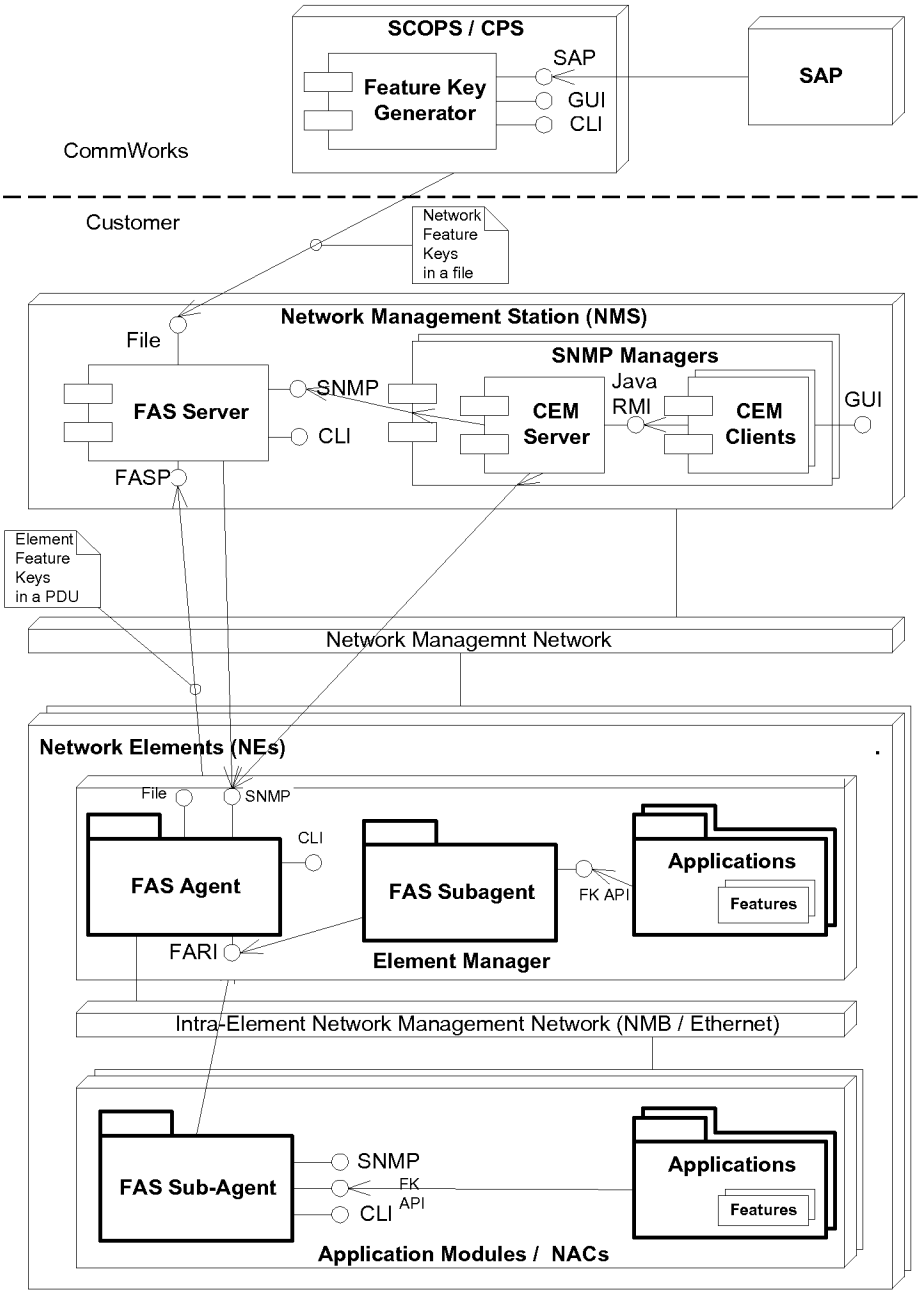
Number	Title
Q0002D	Carrier System Manual Part 4: DESIGN CONTROL
E0129	Glossary of Terms for Product/Technology Development
3G 3.0 FFD #56	CommWorks Wireless 3G NextGen 1.0 Feature Functional Description (FFD) for Feature # 56
F0056/S0053	Product Requirements Definition
E0146	System Functional Specification Procedure
E0130	Peer Review Procedure
E0096	Engineering Management Team Guidelines
v 1.2, XXXXXXX	/Enhanced FEK/FAS 30142 SySFS
v 1.0, XXXXXXX	/Enhanced FEK/FAS Sub-Agent FS
v 1.0, XXXXXXX	/Enhanced FEK/FAS Agent Sw FS

5 System Functional Overview

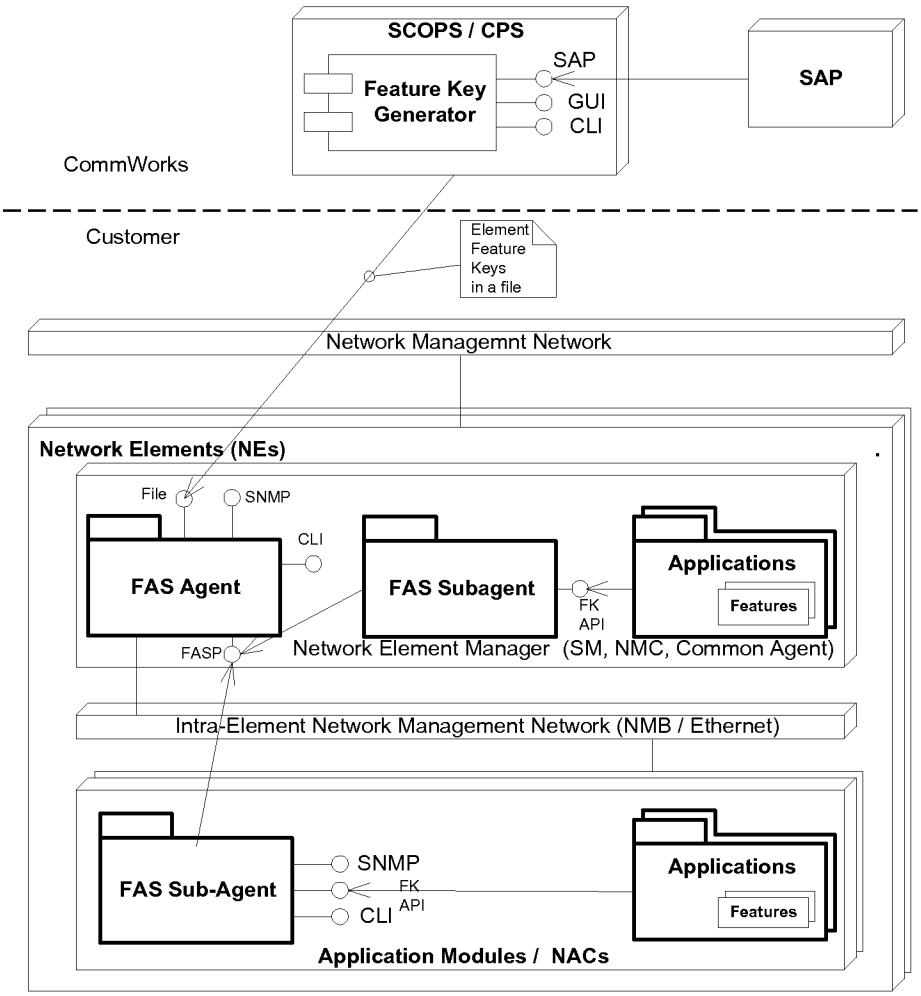
This section should provide a detailed functional overview of the feature. Most likely, this section would contain the same information as that contained in the functional specification.

5.1 Architecture

FAS deployment diagram with a FAS Server



FAS deployment diagram without a FAS Server

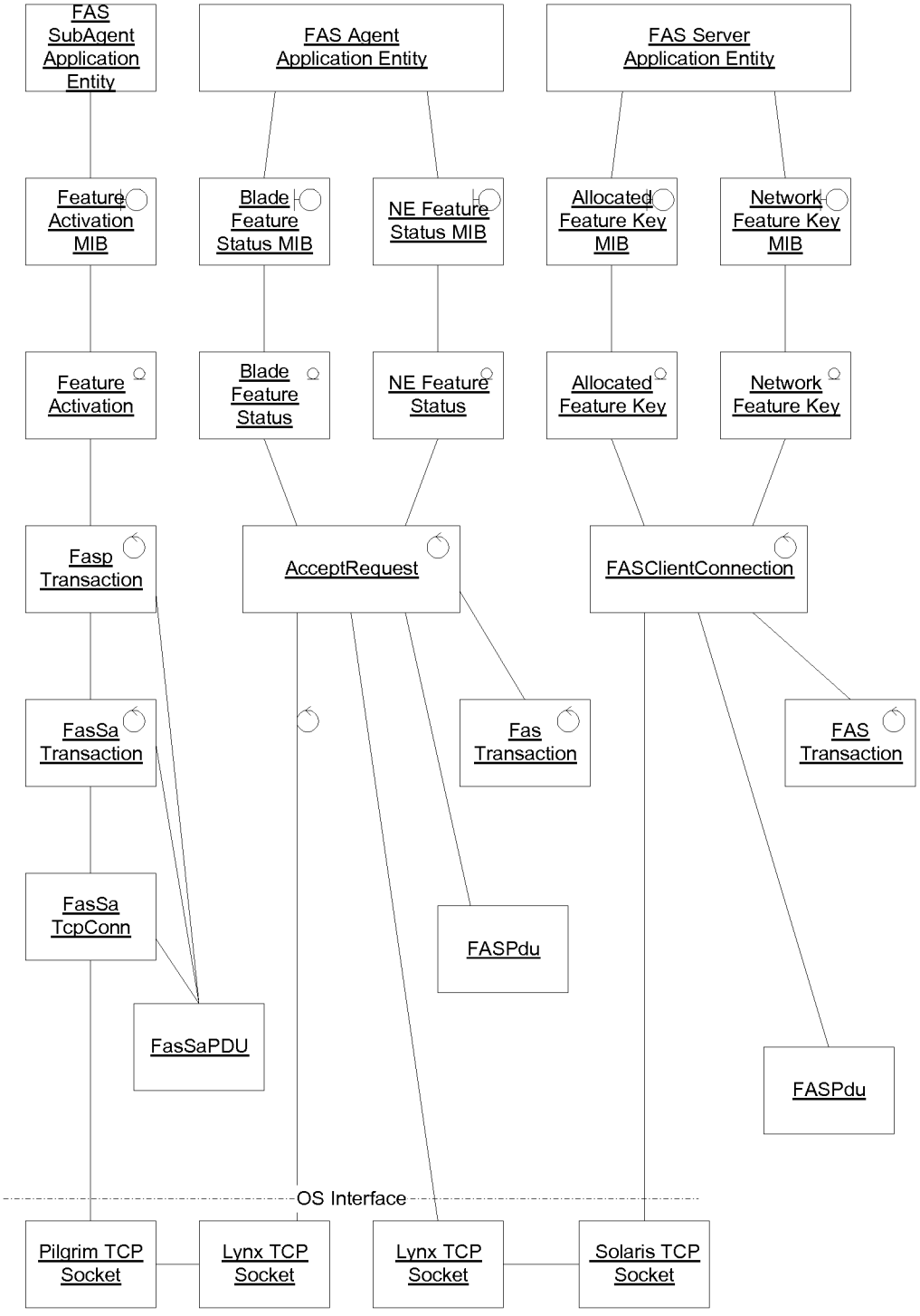


6 High-Level Design

The section will identify and describe the architecture and the feature components of the system in a very high-level fashion.

6.1 FAS Layering

FAS Layering



The naming of entities and objects is based on who is providing the service.

In the FAS SubAgent - FAS Agent relationship, the FAS Agent is the entity providing service to the FAS SubAgent.

In the FAS Agent - FAS Server relationship, the FAS Server is the entity providing the service to the FAS Agent.

7 Detailed Design

The section will contain a detailed description of the components that make up the feature. This could be a detailed description of the State Machine design, Data Flow Diagram, Event Sequence Diagram, Object Modeling, or Pseudo Codes, etc. A combination of the above design techniques should be used.

7.1 Applications Program Interfaces

The section will contain a detailed description of the application program interfaces defined for this feature (if applicable).

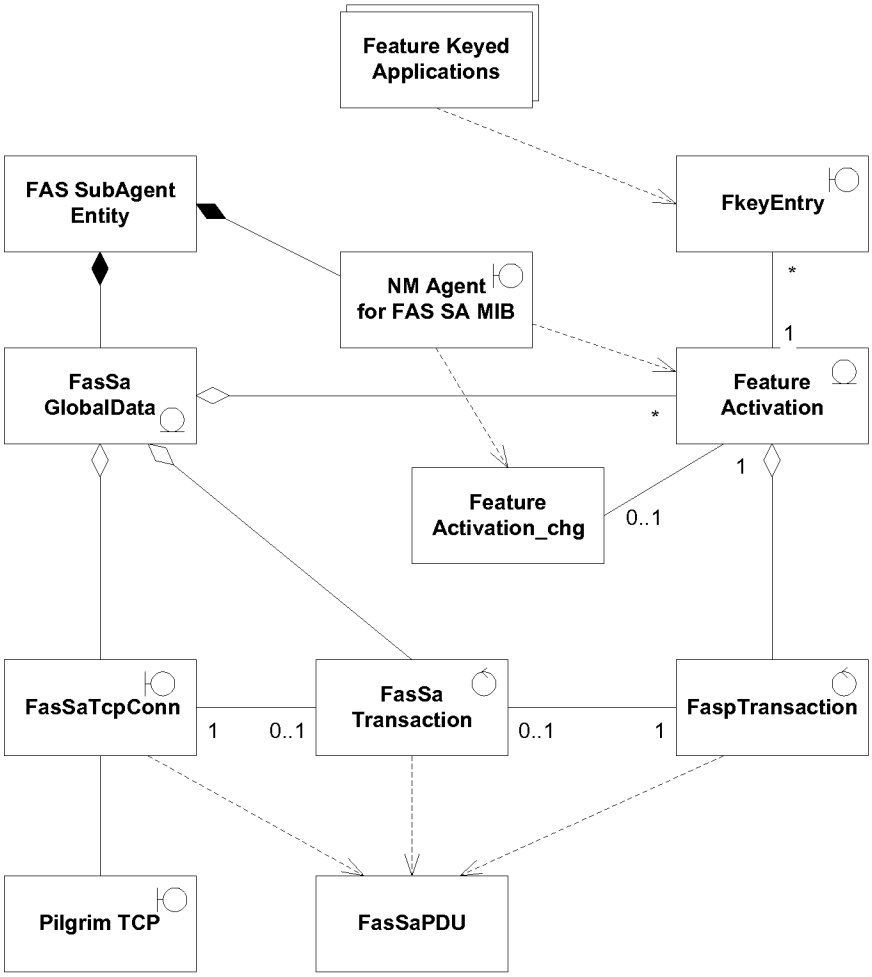
See FkeyEntry object description below for details.

7.2 Major Data Structures

7.2.1 FAS SubAgent Application Entity

This object represents the FAS Entity that resides in blades

FAS SubAgent class diagram



7.3 Behavior

7.3.1 Fas Subagent Initialization

Config process interacts with RoboExec to create and initialize the FAS Subagent Pilgrim process.

The fassa_init routine initializes the fassa_data : FasSaGlobalData structure. It creates sends a mesage to NM to register the USR-FAS-SA-MIB.

When the NMg_REGISTER_RSP message is received from NM, the FAS Subagent will create a row for each featureID that is possible on that blade. This in turn interacts with fkey_api to find out which features are present on the blade. When the FeatureActivation rows are created, the FAS Subagent will send a request to CFM to load the FeatureActivation table from CFM .cfg file.

CFM will perform a SET on --CONFIGURABLE MIB object for every stored row. This will cause a SET to row status to enable each feature that has been saved as enabled.

When the CFM sends a message that CFM Load is complete, the FAS Subagent will start a FaspTransaction for every enabled feature. Each FaspTransaction will be in Fasp_state_pending.

A TCP connection will be opened to the FAS Agent.

Once opened, the FAS Subagent will pick the first FeatureActivation entry with a pending FaspTransaction, start a FasSaTransaction with a unique transaction id, send a FAS_FEATURE_REQUEST PDU to the FAS Agent via the FasSaTcpConn object, and move the FaspTransaction to the Fasp_state_req_sent state.

The FAS Agent will allocate or verify the feature allocation to that blade identified by its entity id of shelf#:slot#, and respond with a positive FAS_FEATURE_RESPONSE

The FAS Subagent validates the response, commit the feature activation, and send a FAS_FEATURE_ACK message back to the FAS Agent to indicate transaction commit.

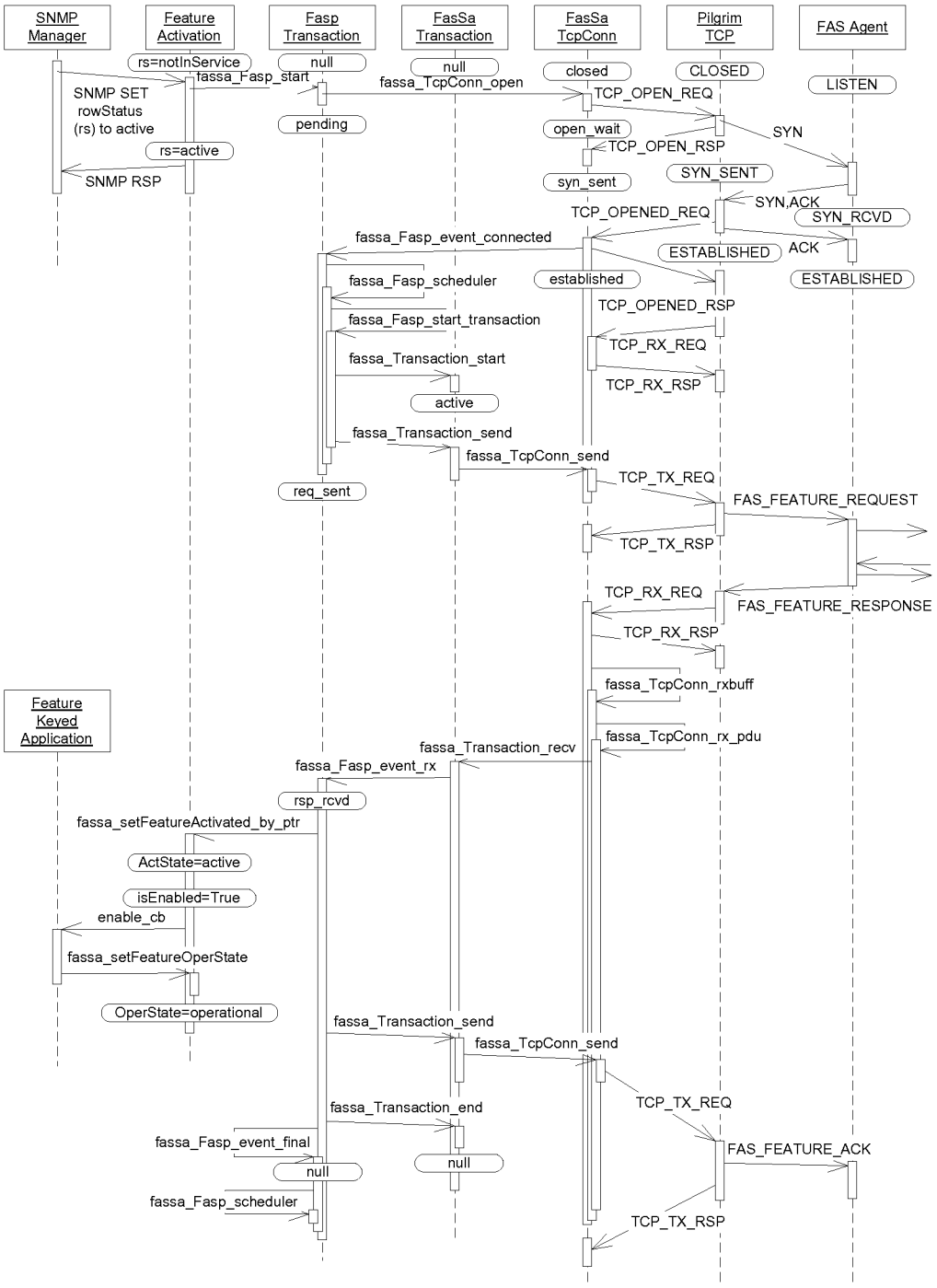
The FAS Subagent completes the FasSaTransaction and FaspTransaction.

The FAS Subagent picks the next FeatureActivation entry with a pending FaspTransaction, start a FasFaTransaction with a fresh transaction_id, and repeat these steps until all activated features have been processed.

When there are no more FaspTransactions to be performed, the Fas Subagent will time-out after 5 seconds, and close the TCP connection.

7.3.2 Feature Activation

Feature Activation Sequence Diagram

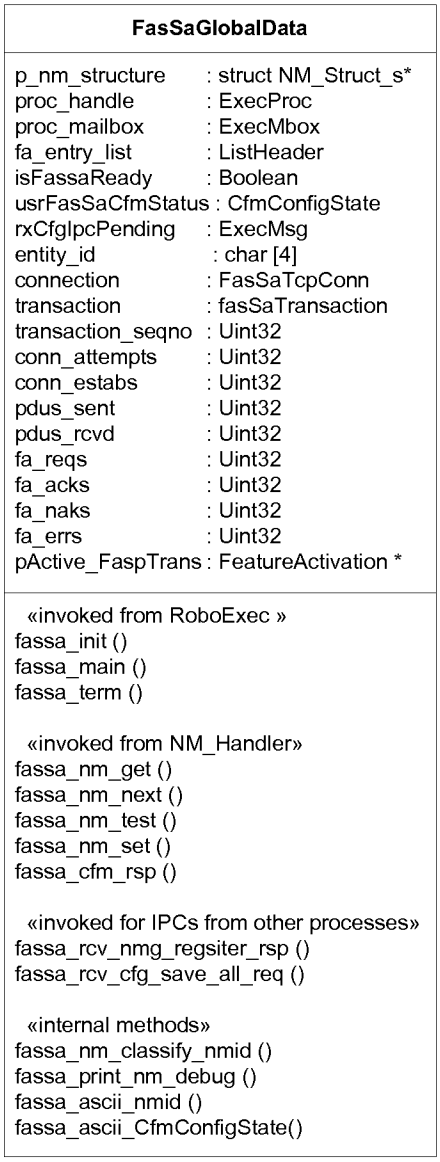


7.4 Objects

7.4.1 FasSaGlobalData

This object is the anchor point for all data structures associated with the FAS Sub-Agent. There is only one instance of this class. The instance is a static global named "fassa_data". It contains the list header of the FeatureActivation

entry list. It contains the single instance of FasSaTcpConn used to communicate with the FAS Agent. It contains only one single instance of FasSaTransaction because FaspTransactions are serialized: only one FaspTransaction at at time is allowed to communicate with the FAS Agent. It contains the entity ID (shelf:slot) that identifies this instance of the FAS Subagent to the FAS Agent.



fassa_init() is invoked by RoboExec when "FAS Subagent" task is created.

fassa_main() is invoked by RoboExec for "FAS Subagent" task to process a IPC message.

fassa_term() would be invoked by RoboExec if "FAS Subagent" task was terminated, which doesn't happen

7.4.2 FkeyEntry

The FkeyEntry object represents one individual activatable feature in the blade. There can be multiple entries per feature set. All features (entries) belonging to the same feature set will have the same featureID.

For example, on the TC2K PDSN, there are 4 instances of FkeyEntry for FeatureID 2:

one instance of FkeyEntry for FKEY_FEATURE_PDSN_PREPAID_AND_HOTLINE,

one instance of FkeyEntry for FKEY_FEATURE_PDSN_IMSI_BASED_AUTH,

one instance of FkeyEntry for FKEY_FEATURE_PDSN_FACN_REDUNDANCY, and

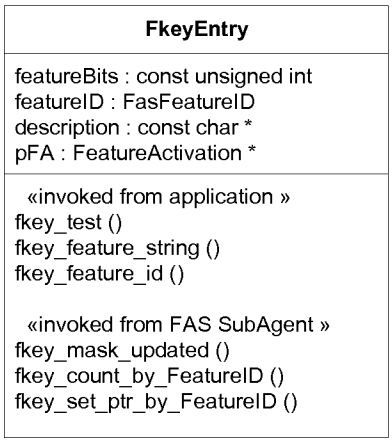
one instance of FkeyEntry for FKEY_FEATURE_PDSN_HA_IPSEC.

On the TC2K HA, there are 2 instances of FkeyEntry for FeatureID 2:

one instance of FkeyEntry for FKEY_FEATURE_PDSN_HA_IPSEC, and

one instance of FkeyEntry for FKEY_FEATURE_HA_VLAN_TAG.

These FkeyEntry objects are contained in vector fkey_table indexed by the FKEY_FEATURE number for testing efficiency at run-time.



The *featureBits* attribute is used for backwards compatibility with the TC1K feature bit system. It contains a single bit set out of 32 bits that will be tested against the TC1K NMC Feature mask. If the Feature mask bit has the corresponding bit set, FAS Subagent will take this to mean that the feature is locally activated and does not need a FAS Feature Key. In TC2K test loads and in sparc testbed, a developer or tester can use the hidden command _SET NMC FEATURE_MASK to activate the feature without a FAS Agent.

The *featureID* attribute is used to associate a FKEY_FEATURE (tested by software function) with a FeatureID (activatable a TC2K FAS Feature key).

The *description* attribute is a text string that is used in 2 places: a) it is displayed in the _SHOW NMC FEATURE_MASK as the description, and b) it is appended to the FeatureActivation's *usrFasFaDescription* attribute.

The *pFA* attribute is set at run-time so that fkey_test() can find the associated FeatureActivation entry to test if feature is permitted or not.

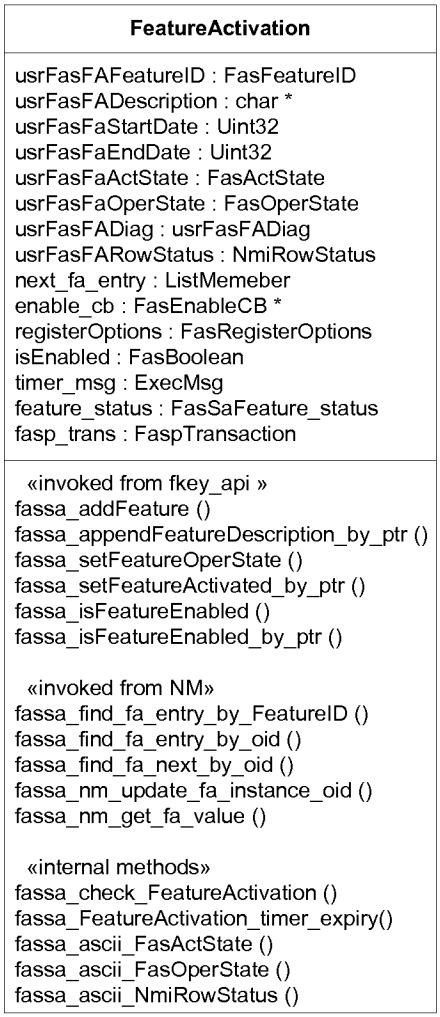
The *fkey_test()* function is used by applications to test whether a feature is enabled (permitted) or not.

The *fkey_feature_id()* function may be used by applications to find out the FeatureID that corresponds to a FKEY_FEATURE value. This is for applications that want to notify the FAS SubAgent when the feature actually becomes Operational.

The *fkey_feature_string()* function is used by CLI and fassa_addFeature to access the description text corresponding to a FKEY_FEATURE value.

7.4.3 FeatureActivation

The FeatureActivation object represents one activatable FeatureID on the blade. One instance can have multiple associated FKEY_FEATUREs. For example, Feature Set 2 is represented as one instance of this object with FeatureID = 2.



The FeatureActivation object contains an attribute for each columnar object that is part of the usrFasFaEntry in the MIB. The value may be an internal representation of the MIB object, or may be the same value.

usrFasFAFeatureID, usrFasFADescription, usrFasFaActState, usrFasFaOperState, usrFasFaDiag, and usrFasFaRowStatus have internally the same value as the external format.

usrFasSaStartDate and usrFasSaEndDate use an unsigned integer count of seconds since 01-JAN-1970 00:00:00 UTC. Leap seconds are not counted. Value 0 means "NULL" i.e. no date. The external format is an 11 octet yymdhmstshm (DateAndTime as per RFC 2579). 01-JAN-0000 00:00:00 UTC indicates "NULL" date.

The *next_fa_entry* attribute is used to link FeatureActivation entries sorted by FeatureID.

The *enable_cb* is a pointer to a function that is invoked when a FeatureActviation entry transitions from disabled (permission = no) to enabled (permission = yes), or vice-versa. The intent is to notify the application at run-time that a feature just got enabled or just got disabled so application can take appropriate actions. Applications that make use of this facility need to invoke *fassa_setFeatureOperState()* when feature actually becomes Operational, or actually becomes Inop.

The *registerOptions* contains the options that was passed in the fassa_addFeature. The options bits are: FKEY_FA_ENABLED - Initialize row as active (want feature), and FKEY_FA_BASELINED when feature is baselined (permission is hardcoded to on).

The *isEnabled* attribute remembers the last feature enable state that was communicated to the application.

The *timer_msg* contains a pointer to the timer message that is currently running. It is NULL if there is no timer running.

The *feature_status* attribute is a bit mask with 1 bit per potential contributor of feature permission: baseline, feature bits, and FAS feature key. Bit is set when a feature when enabled by the corresponding source.

The *fasp_trans* : FaspTransaction object is contained in the FeatureActivation object for convenience to avoid having to dynamically allocate memory to perform a FaspTransaction, and free it afterwards. It a FasoTransaction conceptually constructed when a fassa_Fasp_start() is called. It is conceptually destroyed when transaction ends (fassa_Fasp_event_final ()).

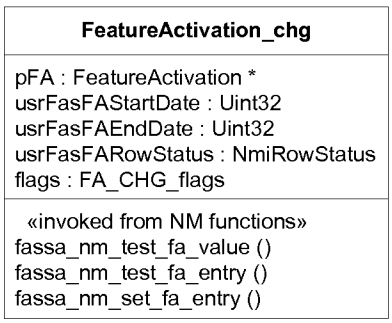
7.4.4 FeatureActivation_chg

The FeatureActivation_chg object is a transient object created while processing a NM SET request. It is used to accumulate tested VARBIND values for a single FeatureActivation row.

It is created when the first varbind of a FeatureActivation row is encountered in the test routine. All varbinds of the same row are processed. It is linked to the varbind.

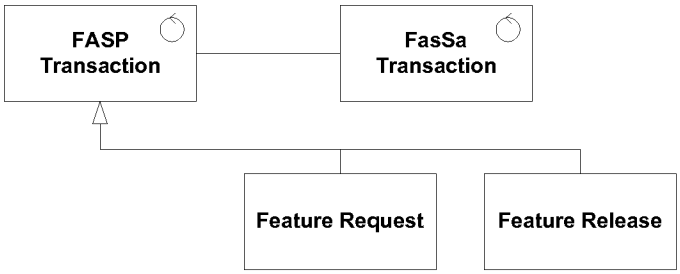
During set (commit) phase NM SET pdu processing, the values from this object are committed to the associated FeatureActivation object.

The NM_Handler function is responsible for freeing all allocated FeatureActivation_chg objects.

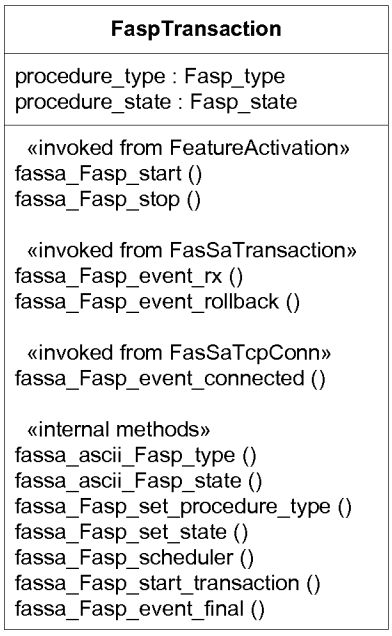


7.4.5 FaspTransaction

The FASP Transaction is a control object. It is transient object that is exists for the duration of the transaction. It contains the sequencing logic for the activation, deactivation, and verification of Feature Keys. It communicates with multiple other objects to realize that logic. In particular, it interacts with a FasSa Transaction object which handles the transaction semantics



The FAS SubAgent Feature Request or Release Procedure is initiated by a Feature Activation entry when a SNMP Set is made to one of its writable MIB objects. It is responsible for updating the Feature Activation entry consistently.



The *procedure_type* attribute is Fasp_type_none when there is no transaction in progress.

It is set to Fasp_type_request to start a transaction to acquire or verify possession of a feature key (permission) for that FeatureID from the FAS Agent. It is set to Fasp_type_release to start a transaction to revoke a FAS feature key back to the FAS Agent.

The *procedure_state* attribute is the state variable of the FASP transaction protocol.

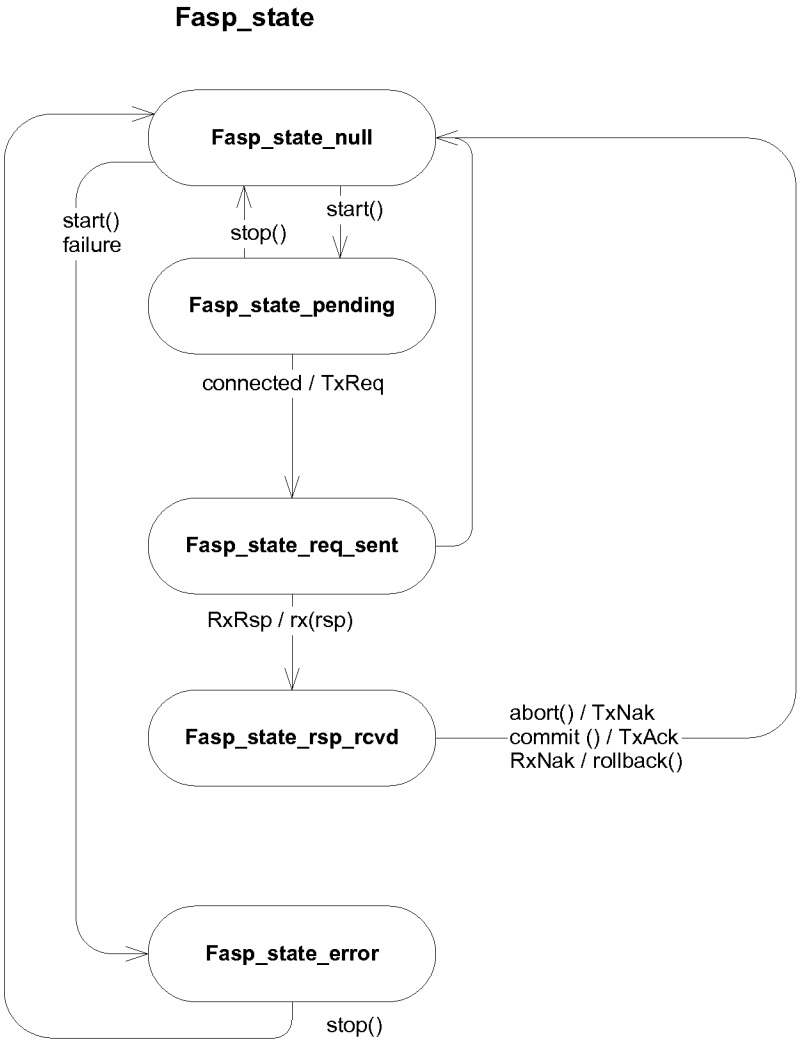
Fasp_state_null: no transaction in progress / transaction complete.

Fasp_state_pending: Transaction is waiting for a TCP Connection or waiting for its turn to use the TCP connection to the FAS Agent

Fasp_state_req_sent: FAS_FEATURE_REQUEST PDU sent

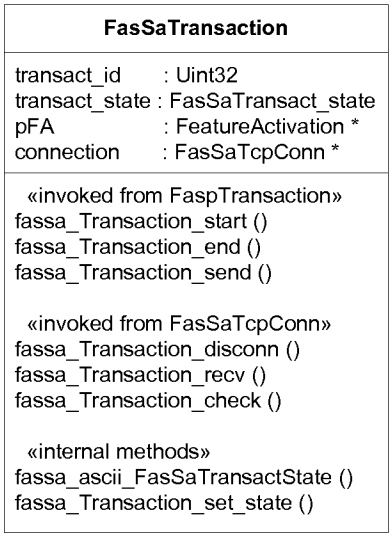
Fasp_state_rsp_rcvd: FAS_FEATURE_RESPONSE PDU received, ACK/NAK not yet sent.

Fasp_state_error: internal failure occurred when starting transaction.



7.4.6 FasSaTransaction

The FasSaTransaction handles one transaction context.



The *transact_id* attribute contains the transaction ID of the current transaction. Response packets received with the wrong transaction ID will be NAKed without changing the state of the current transaction.

The *transact_state* attribute contains the transaction's state:

FasSaTransact_state_null - transaction not in progress

FasSaTransact_state_active - transaction in normal progress state

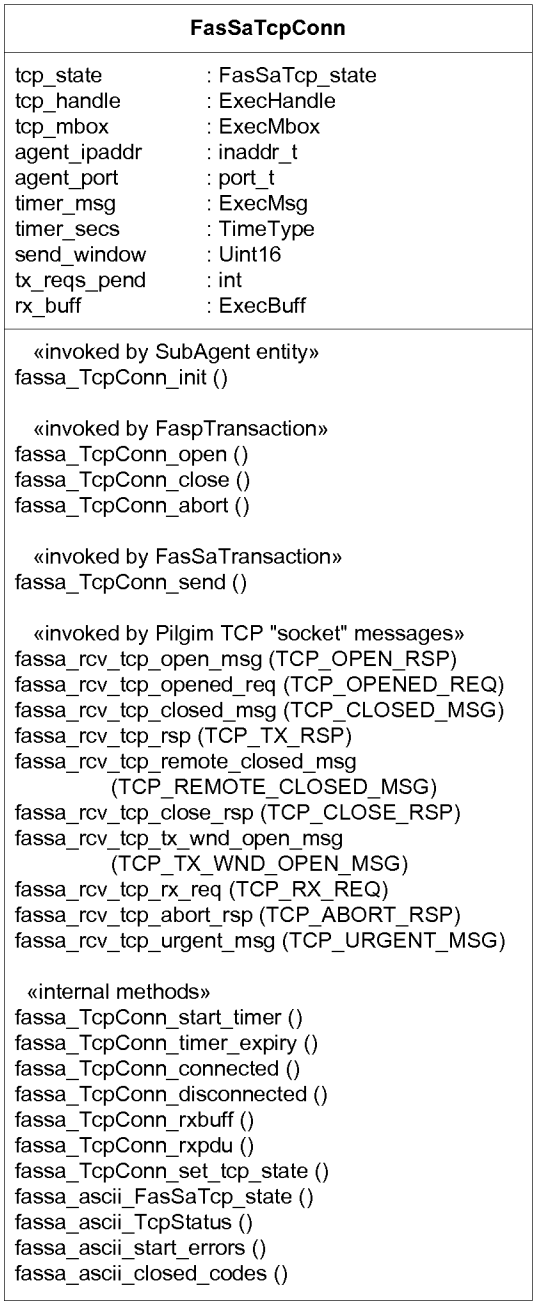
FasSaTransact_state_rollback - transaction was aborted. Waiting for FaspTransaction rollback to complete.

The *pFA* attribute indicates the FeatureActivation entry associated with the current transaction.

The *connction* attribute indicates which TCP connection is associated with the current transaction.

7.4.7 FasSaTcpConn - the FAS Pilgrim TCP Socket I/F

This object mediates between the Pilgrim TCP socket and the other FAS SubAgent objects. It keeps track of the asynchronous communication with Pilgrim TCP.



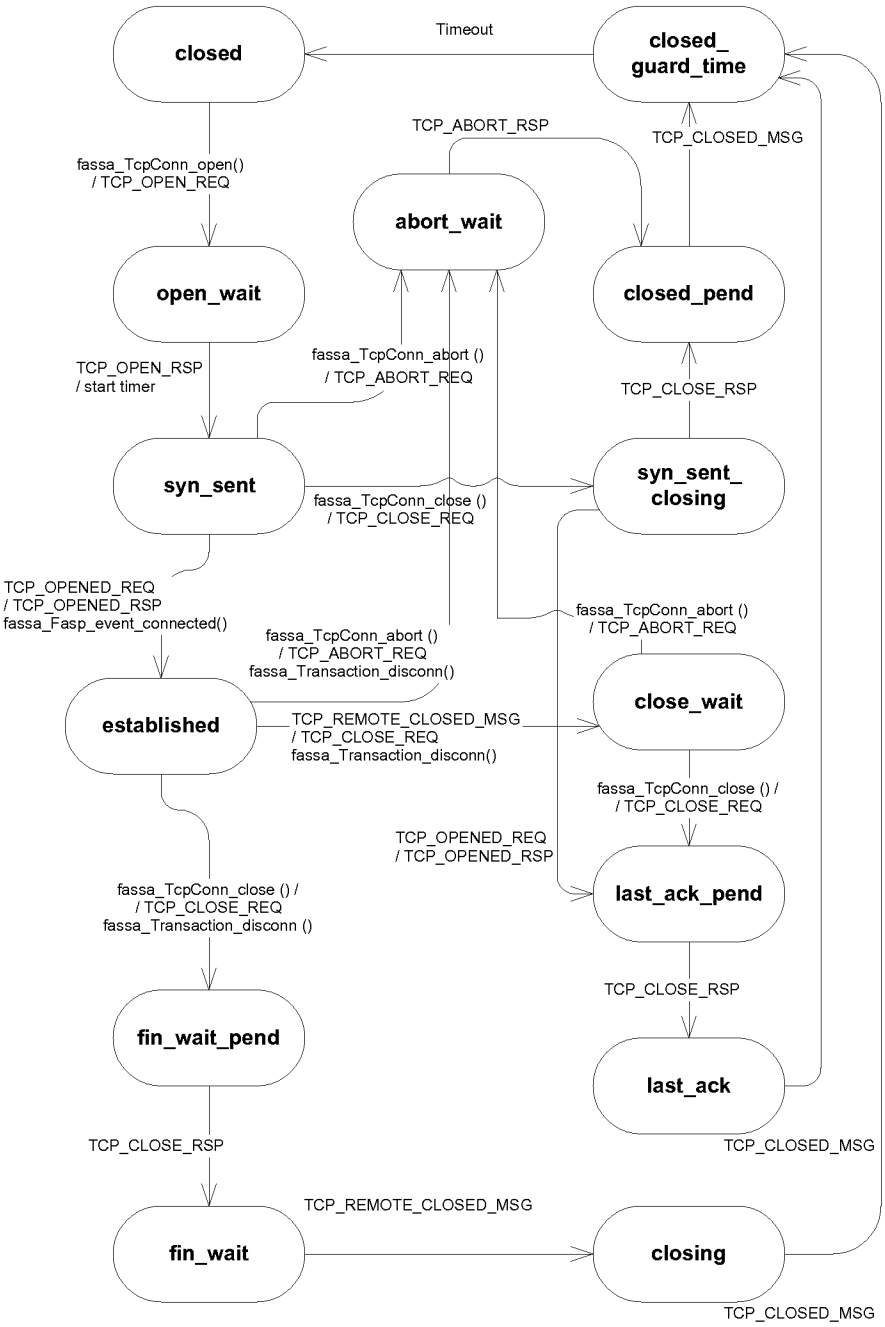
The *tcp_handle* is a Pilgrim handle to the Epilogue TCB (TCP Control Block) for the current TCP connection. There is a new TCB for every new TCP connection. It is placed in messages going to Pilgrim's TCP process so it knows which TCP context to use. This task validates the TCB to make sure that the TCP message is fresh (applies to the current connection) rather than belated messages that come from a lingering TCP context.

The *agent_ipaddr* TCP destination address for the FAS Agent. It is automatically determined by looking at the VxWorks parsed boot line which contains where the blade was booted from - the System Manager.

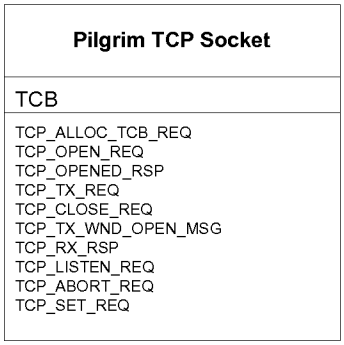
XXXXXXXXXXXXXXXXXXXXX

The *rx_buff* attribute points to a Buffer Chain to accumulate packet segments to form one complete PDU before parsing it.

tcp_state Finite State Machine



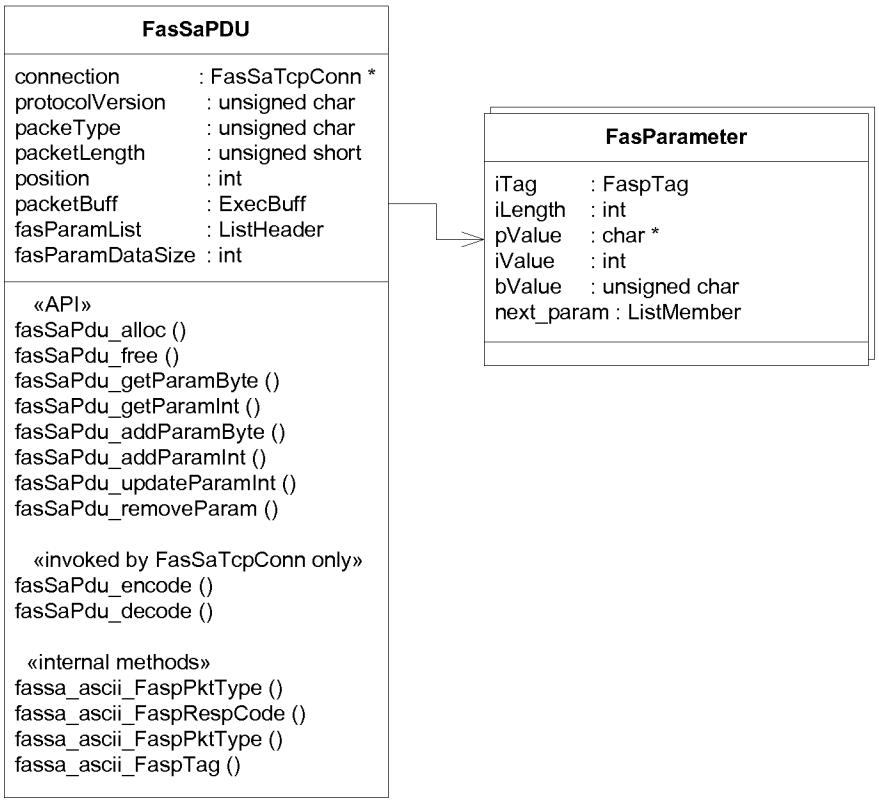
7.4.8 Pilgrim TCP "socket" interface



7.4.9 FasSaPDU

This object is used to handle TLV encoding of FASP PDUs. PDUs are initially constructed with only a packetType and protocolVersion. Each Parameter is added by a separate function all, which results in a FasParameter object to be added to a link list anchored in the FasSaPDU. When PDU is logically complete, it is sent to the FasSaTcpConn object which will invoke the fasSaPdu_encode() function to allocate a Pilgrim ExecBuff, build it from the FasSaPDU contents and associated FasParameters including a computed checksum.

For receive, when a complete PDU has been received in a buffer chain, a FasSaPDU is allocated, the buffer chain is given to that PDU, and fasSaPdu_decode() function is invoked to build the linked list of FasParameter objects.



FasParameter entries are added using fasSaPdu_addParamByte() and fasSaPdu_addParamInt() functions with the TAG that identifies the parameter.

FasParameter entries are accessed using fasSaPdu_getParamByte() and fasSaPdu_getParamInt() functions providing the TAG that identifies the parameter to be retrieved. An error is returned if the parameter is missing.

There are 5 packet types:

The FAS_FEATURE_REQUEST packet:

- 4 byte header with protocolVersion = 1 and packetType = FAS_FEATURE_REQUEST

- byte parameter PACKET_SUBTYPE = FAS_SOURCE_SUBAGENT

- int parameter TRANSACTION_ID = transaction ID

- int parameter ENTITY_ID = "a:bb" where a = shelf 1..6, and bb = slot 01..17

- int parameter FEATURE_IDENTIFIER = featureID

- int parameter TOTAL_FEATURE_UNITS = 0 or 1

- int parameter PACKET_CHECKSUM

The FAS_FEATURE_RESPONSE packet:

- 4 byte header with protocolVersion = 1 and packetType = FAS_FEATURE_RESPONSE

- int parameter PACKET_CHECKSUM

- byte parameter PACKET_SUBTYPE = FAS_SOURCE_SUBAGENT

- int parameter TRANSACTION_ID = transaction ID

- int parameter FEATURE_IDENTIFIER = featureID

- int parameter TOTAL_FEATURE_UNITS = 0 or 1

- int parameter PACKET_RESPONSE_CODE = return code

The FAS_FEATURE_ACK packet:

- 4 byte header with protocolVersion = 1 and packetType = FAS_FEATURE_ACK

- byte parameter PACKET_SUBTYPE = FAS_SOURCE_SUBAGENT

- int parameter TRANSACTION_ID = transaction ID

- int parameter PACKET_CHECKSUM

The FAS_FEATURE_NAK packet:

- 4 byte header with protocolVersion = 1 and packetType = FAS_FEATURE_NAK

- byte parameter PACKET_SUBTYPE = FAS_SOURCE_SUBAGENT

- int parameter TRANSACTION_ID = transaction ID

- int parameter PACKET_CHECKSUM

The FAS_FEATURE_TERMINATE packet:

- 4 byte header with protocolVersion = 1 and packetType = FAS_FEATURE_TERMINATE

7.5 Error Handling

7.5.1 Error Handling Pattern Rationale

A large portion of source code deals with run-time errors such as invalid function arguments, invalid message parameters, failures from called functions, and so on. This is done to make the product robust because a failure could affect hundreds of users, and to provide some indication that can be used by maintenance staff for troubleshooting.

First, lets distinguish processing errors (i.e. bugs in the code, out-of-memory), from communications errors such as call failures, protocol violations, etc., and user errors.. Communication, and user errors can happen even with perfect code. They are abnormal situations which are part of specified behavior. These errors sometimes need to be logged, and usually have specified protocol responses (i.e. retries, resets, call clearing, etc.). Processing errors, on the other hand, contradict specified behavior, they are not supposed to happen.

The handling of processing errors have some common requirements. The errors should be detected as early as possible. Useful information should be collected to help the developers analyze the error. The information should be concise. Resources should be cleaned-up. The system state should be put in a as consistent state as practicable. Errors should be contained in a way to affect as few users as possible. Handling of processing errors should be clearly distinguished from handling of expected behaviour.

A fault in a software system can cause one or more errors. The chain of errors and failures caused by a single fault is called error propagation.

The error handling code should be separated from normal code. Without any separation the normal code will be cluttered by a lot of error handling code. This makes code less readable, error prone and more difficult to maintain.

This pattern allows developers concentrate on domain code rather than technical error handling stuff. It makes error handling consistent. Changes to normal code does not need a cascade of changes to the error handling code. By making error checking easy, errors are more likely to be detected early and limit their damage.

7.5.2 Error Handling Pattern

The error handling is clearly separated from the normal code, with the normal code first. The normal path is straight-line code rather than deeply nested if statements. This makes the source code of the function have the same layout as a use-case which also has the normal path first. Note that both normal and alternative paths of a use-case are considered normal code. Error handling is for internal code faults. For background information, see "Error Handling for Business Information Systems".

NOTE that FAS_ASSERT mechanism DOES NOT crash the box. On the contrary, the mechanism goes to great lengths to avoid crashes, to do best-effort cleanup so the code can continue normal processing.

Nearly every function will follow the same pattern:

```
ReturnCode everyFunction (args)

{

      FAS_ERR_INIT ("everyFunction");
```

```
/* local variable declarations */
ExecStatus exStatus;
ReturnCode returnCode;

/* normal code */
FAS_TRY
{
      /* arg checks */
      FAS_ASSERT_ARG (arg check, "arg name", "expected value");

      /* pointer checks */
      FAS_ASSERT_PTR (ptr check, "ptrName");

      /* pre-condition checks */
      FAS_ASSERT_PRECOND (precond check, "precond", "expected precond");

      /* state checks */
      FAS_ASSERT_STATE (state check, "stateVar", "expected states");

      /* result checks */
      result = someFunction ();
      FAS_ASSERT_RESULT (result ok, "someFunction", "expected result");

      exStatus = somePilgrimFunction (args);
      FAS_ASSERT_EXEC_SUCCESS (exStatus, "somePilgrimFunction");

      /* more success path code */
      /* . . . */
```

```
            /* post-condition checks */

            FAS_ASSERT_POSTCOND (postcond check, "postcond", "expected
postcond");


            returnCode = SUCCESS;

      }


      /* error handling code */

      FAS_CATCH ()

      {

            FAS_DEFAULT_CATCH_ACTION (); /* prints msg from FAS_ASSERT */


            /* print args unless you know a higher level function will print
them */


            /* print context including state unless you know a higher level
fuction

       will print them */


            /* cleanup code: free unsent messages, partially constructed
structures,

       put variables in a consistent state, put objects in error state, etc.*/


            returnCode = FAIL;

      }


      return returnCode;

}
```

7.5.3 Error Logging: Error Propagation and Backtrace

The error handling code is responsible for adding context information such as instance and state to the event log. The fact that the context information is added separately to the event log from the error information allows the error

information generation to be very generic. This in turn allows the error detection code to be less visually intrusive in the normal code path which in turn allows the normal code path to more readable and less error prone. Functions to make ascii strings out of state variables and object instances make it easy to log this information.

The failure return code of the function causes the calling function to FAS_ASSERT also, triggering its own error logging and cleanup. This error logging, propagation, and stack unwinding continues until the main event handler is reached, at which point the error propagation is intentionally ended. This error propagation has a side-effect of writing a backtrace of the call stack into the event log from the asserting function all the way back up to the main event handler. This is intentional, as the call stack and object context is very useful in error analysis. However, due to the volume of data written, it is important to use the FAS_ASSERT mechanism only for code faults, and not for user errors, communication failures or remote protocol violations.

7.5.4 Processing Fault Error Detection (Error Traps)

For error detection, the code must be enriched with a number of run-time checks. The state and the behavior of the system is verified in nearly every function.

Invalid parameters need to be checked on entry of the function.

Violation of the pre-condition needs to be checked at the entry of the function.

Unexpected results or failures from functions needs to be tested immediately after the return of the function.

Function invariants need to be checked at appropriate places so they won't cause undue overhead.

Violation of the post-condition needs to be checked at the end of the normal path of the function.

7.5.5 Exception Handling Simulation

The FAS_ERR_INIT("functionName') is a macro to define local variables for error handling, assign the function name to the "function" local variable, and initializes "errLine" local variable to 0;

The FAS_TRY {} FAS_CATCH () {} is a poor-man's exception simulation. This translates into:

```
do                  /* FAS_TRY */

{

}



while (0);          /* FAS_CATCH */

if (errLine > 0)    /* FAS_CATCH */

{

}
```

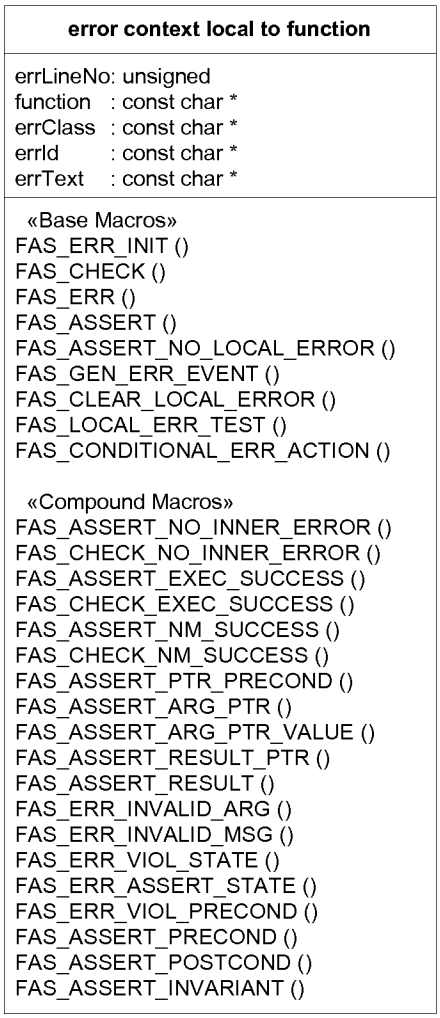
The FAS_ASSERT_* macros capture the line number of the error, an error class, an error id, some error text by assigning them to local variables defined by FAS_ERR_INIT macro, and cause a "break" out of the FAS_TRY (do while) block. The FAS_CATCH () defines the end the FAS_TRY block, checks for an error, and if so, performs the error handling block. The FAS_DEFAULT_CATCH_ACTION() captures the function name from the FAS_ERR_INIT, the file name, the FAS_ASSERT_ information, into an error object and causes that error object to be formatted to the event log.

The FAS_ASSERT_* macros together with the FAS_DEFAULT_CATCH_ACTION macro captures localization information of the exact place where the error was detected. The errors are structured into logical groups (error class) so the errors could be handled by category, and to cut down on programmer time in inventing new error messages.

7.5.6 Resource Cleanup

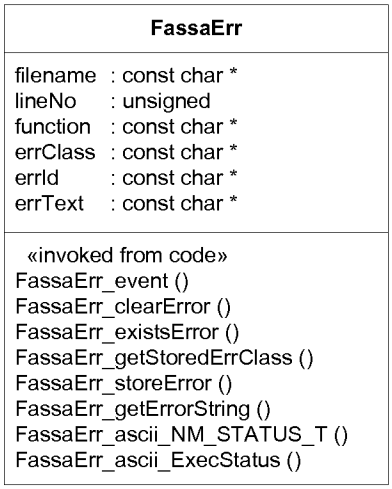
The normal path sets variables in such a way the error handling code can figure out what to cleanup. For example: the pointer to a message being built is initialized to 0, is set when the message is allocated, and is cleared the moment the message is sent. The cleanup code will free the message if the pointer is not null. If an FAS_ASSERT error occurs before the message is allocated, the pointer is null, and there is nothing to be freed. If an FAS_ASSERT error occurs after the message is allocated, but not sent, the message pointer is not null, and the message will be freed. If an FAS_ASSERT error occurs after the message is sent, the pointer is null again, and nothing needs to be freed.

7.5.7 error context local to function



7.5.8 FassaErr

Global object used to simulate exceptions in C code. An error is stored in this object. The calling function can query this object to see if an inner function encountered a simulated exception. This is used to check if the error is a fresh error or propagated error (failing function causes outer function to fail in a error cascade until the propagation is intentionally stopped. This occurs at the top level function (eg fassa_main).



7.6 Modules impacted

The existing modules that are impacted by this feature should be listed in this section (if applicable)

/pilgrimvobs/bsp/gwc_ppc/application/make_template.mk

/pilgrimvobs/common/NetMan/CLI/include

/pilgrimvobs/common/NetMan/CLI/src

/pilgrimvobs/common/mibs

/pilgrimvobs/common/NetMan/CLI/include/CLI.str

/pilgrimvobs/common/NetMan/CLI/include/cli_strings.h

/pilgrimvobs/common/NetMan/CLI/include/cli_verify.h

/pilgrimvobs/common/NetMan/CLI/src/cli.c

/pilgrimvobs/common/NetMan/CLI/src/cli_chassis.c

/pilgrimvobs/common/NetMan/CLI/src/cli_core.hin

/pilgrimvobs/common/NetMan/CLI/src/cli_utils.c

/pilgrimvobs/common/NetMan/CLI/src/cli_verify.c

/pilgrimvobs/common/NetMan/NM-API/include/nm_func.h

/pilgrimvobs/common/mibs/makefile

/pilgrimvobs/common/mibs/usr_definitions.mib

/pilgrimvobs/common/NetMan/NM-API/include/fkey_api.h

/pilgrimvobs/common/NetMan/NM-API/src/fkey_api.c

/pilgrimvobs/common/mibs/usr_fas_sa.mib

/pilgrimvobs/common/NetMan/CLI/include/cli_fassa.h

/pilgrimvobs/common/NetMan/CLI/src/cli_fassa_ids.c

/pilgrimvobs/common/NetMan/CLI/src/cli_fassa.c

/pilgrimvobs/common/NetMan/CLI/src/cli_fassa.hin

/pilgrimvobs/pilgrim/.

/pilgrimvobs/pilgrim/config/src/config_NETSERVERPPCGWC.c

/pilgrimvobs/pilgrim/config/src/config_TESTBED.c

/pilgrimvobs/pilgrim/include/facility.str

/pilgrimvobs/pilgrim/include/pilgrim.h

/pilgrimvobs/pilgrim/include/pilgrim_procs.h

/pilgrimvobs/pilgrim/makefile

/pilgrimvobs/pilgrim/testbed/sparc/makefile

/pilgrimvobs/pilgrim/testbed/src/PilgrimStrings

/pilgrimvobs/pilgrim/testbed/src/main.c

/pilgrimvobs/pilgrim/fas

/pilgrimvobs/pilgrim/fas/include

/pilgrimvobs/pilgrim/fas/src

/pilgrimvobs/pilgrim/fas/obj

/pilgrimvobs/pilgrim/fas/obj/sparc

/pilgrimvobs/pilgrim/fas/obj/sparcD

/pilgrimvobs/pilgrim/fas/obj/ppcG

/pilgrimvobs/pilgrim/fas/obj/ppcGD

/pilgrimvobs/pilgrim/fas/include/makefile

/pilgrimvobs/pilgrim/fas/include/fassa_err.h

/pilgrimvobs/pilgrim/fas/include/fassa.str

/pilgrimvobs/pilgrim/fas/include/fassa_pdu.h

/pilgrimvobs/pilgrim/fas/include/fassa.h

/pilgrimvobs/pilgrim/fas/include/fassa_protos.h

/pilgrimvobs/pilgrim/fas/include/fassa_private.h

/pilgrimvobs/pilgrim/fas/include/fassa_mibdefs.h

/pilgrimvobs/pilgrim/fas/src/makefile

/pilgrimvobs/pilgrim/fas/src/fassa_err.c

/pilgrimvobs/pilgrim/fas/src/fassa_nm.c

/pilgrimvobs/pilgrim/fas/src/fassa_tcp.c

/pilgrimvobs/pilgrim/fas/src/fassa_trans.c

/pilgrimvobs/pilgrim/fas/src/fassa_feat.c

/pilgrimvobs/pilgrim/fas/src/fassa_pdu.c

/pilgrimvobs/pilgrim/fas/src/fassa_fasp.c

/pilgrimvobs/pilgrim/fas/src/fassa_main.c

/pilgrimvobs/pilgrim/fas/obj/sparc/makefile

/pilgrimvobs/pilgrim/fas/obj/sparcD/makefile

/pilgrimvobs/pilgrim/fas/obj/ppcG/makefile

/pilgrimvobs/pilgrim/fas/obj/ppcGD/makefile

/pilgrimvobs/pilgrim/fas/obj/make_template.mk

/pilgrimvobs/pilgrim/fas/makefile

/pilgrimvobs/platforms/gwc_ppc/CLI/make_platform.mk

/pilgrimvobs/platforms/gwc_ppc/nmbagent/src/nmb_agent_config.c

/pilgrimvobs/platforms/gwc_ppc/nmbagent/src/nmb_agent_main.c

/pilgrimvobs/platforms/testbed/CLI/make_platform.mk

8 RADIUS Attributes Support

none

9 Configuration and Command Line interfaces

This section describes all the configurations and CLI commands for this feature.

LIST FEATURES

SHOW ALL FEATURES

SHOW FEATURE n

ENABLE FEATURE n

DISABLE FEATURE n

SET FEATURE n

[START_DATE dd-mmm-yy[yy][.hh.mm[.ss]]]

[END-DATE dd-mmm-yy[yy][.hh.mm[.ss]]]

[ENABLED {YES|NO}]

10 MIB Definitions

This section contains the MIB definitions in detail for this feature.

```
USR-FAS-SA-MIB DEFINITIONS ::= BEGIN


IMPORTS
   common
      FROM USR-DEFINITIONS-MIB
   Integer32, MODULE-IDENTITY, OBJECT-TYPE
      FROM SNMPv2-SMI
   DisplayString, RowStatus, DateAndTime, TEXTUAL-CONVENTION
      FROM SNMPv2-TC
   OBJECT-GROUP, MODULE-COMPLIANCE
      FROM SNMPv2-CONF;


usrFasSA MODULE-IDENTITY
   LAST-UPDATED " XXXXXXX "
```

```
    ORGANIZATION
        "Commworks Corp, a 3Com company"
    CONTACT-INFO
        "Postal: CommWorks Corporation.
         3800 Golf Road
         Rolling Meadows, Illinois 60008
         US
         Tel:   +1 847-262-5000"
    DESCRIPTION
        "The MIB module to describe objects for the CommWorks
             Feature Activation System (FAS) Subagent.
             The FAS Sub Agent is the subsystem in each application module or
             application blade that controls the activation and deactivation
             of features in that application module."
    REVISION    " XXXXXXX "
    DESCRIPTION
        "Initial revision."
    ::= { common 94 }

usrFasSAObjects     OBJECT IDENTIFIER ::= { usrFasSA 1 }
usrFasSAConformance OBJECT IDENTIFIER ::= { usrFasSA 2 }

usrFasSAGroups      OBJECT IDENTIFIER ::= { usrFasSAConformance 1 }
usrFasSACompliances OBJECT IDENTIFIER ::= { usrFasSAConformance 2 }

-- Textual Conventions

FasFeatureID ::= TEXTUAL-CONVENTION
```

```
    STATUS        current
    DESCRIPTION
            "Uniquely identifies a feature that can be enabled."
    SYNTAX        Integer32 (0..2147483647)

-- The entities modeled in this MIB module are:
--   Feature Activation

usrFasFATable   OBJECT-TYPE
    SYNTAX        SEQUENCE OF UsrFasFAEntry
    MAX-ACCESS          not-accessible
    STATUS        current
    DESCRIPTION
            "The Feature Activation Table."
    ::= { usrFasSAObjects 1 }

usrFasFAEntry OBJECT-TYPE
    SYNTAX        UsrFasFAEntry
    MAX-ACCESS          not-accessible
    STATUS        current
    DESCRIPTION
            "A conceptual row in the Feature Activation table (usrFasFATable).

            Entries are created at initialization time.
            "
    INDEX   { usrFasFAEntityId, usrFasFAFeatureID }
    ::= { usrFasFATable 1 }
```

```
UsrFasFAEntry ::= SEQUENCE {
   usrFasFAEntityId        Integer32,
   usrFasFAFeatureID       FasFeatureID,
   usrFasFADescription DisplayString,
   usrFasFAStartDate       DateAndTime,
   usrFasFAEndDate         DateAndTime,
   usrFasFAActState        INTEGER,
   usrFasFAOperState       INTEGER,
   usrFasFADiag INTEGER,
   usrFasFARowStatus       RowStatus
}

usrFasFAEntityId OBJECT-TYPE
   SYNTAX       Integer32 (0..2147483647)
   MAX-ACCESS          not-accessible
   STATUS       current
   DESCRIPTION
         "Attribute that uniquely identifies the application entity
         Attribute that uniquely identifies the application entity.
         For most applications, it will be 1.  This attribute come into play
         when multiple application entities use the same SNMP agent.
         This occurs in the Common Agent environment where multiple
         applications use the same Common Agent on the same machine, eac
         potentially using their own set of features.  The EntityId is used to
         distinguish the application entities."
   ::= { usrFasFAEntry 1 }

usrFasFAFeatureID OBJECT-TYPE
```

SYNTAX FasFeatureID

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"Attribute that uniquely identifies the row in the table.

It identifies a particuar feature supported by the blade."

::= { usrFasFAEntry 2 }

usrFasFADescription OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Textual description of the feature"

::= { usrFasFAEntry 3 }

usrFasFAStartDate OBJECT-TYPE

SYNTAX DateAndTime

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"Point in time when the permission begins.

When a SET is made to this object, a FAS transaction to the FAS Agent

is initiated, but the GET value does not get updated until the FAS

transaction is successful. If the FAS transaction fails or aborts,

the object GET value remains with its pre-transaction content.

Year 0000 shall indicate that there is no start point."

DEFVAL { '000001010000002B0000'H } -- Jan 1, year 0, 00:00:00 UTC

```
    ::= { usrFasFAEntry 4 }


usrFasFAEndDate OBJECT-TYPE
    SYNTAX      DateAndTime
    MAX-ACCESS          read-create
    STATUS      current
    DESCRIPTION
            "Point in time when the permission ends.
            When a SET is made to this object, a FAS transaction to the FAS Agent
            is initiated, but the GET value does not get updated until the FAS
            transaction is successful.  If the FAS transaction fails or aborts,
            the object GET value remains with its pre-transaction content.  If
            the FAS transaction succeeds as a partial grant then objects gets
            updated with granted end time.  For example, a feature key
            Year 0000 shall indicate that there is no end point."
    DEFVAL      { '000001010000002B0000'H }  -- Jan 1, year 0, 00:00:00 UTC
    ::= { usrFasFAEntry 5 }


usrFasFAActState OBJECT-TYPE
    SYNTAX    INTEGER {
      active  (1),
            inactive (2)
            }
    MAX-ACCESS          read-only
    STATUS      current
    DESCRIPTION
            "This is the activation state of the feature.
            active (1):  indicates that the blade has got a Feature Key for
```

```
                this feature.
        inactive (2): indicates that this blade doesn't have a Feature Key
                for this feature.
        This doesn't mean that the provisioned feature is being used now."
    ::= { usrFasFAEntry 6 }

usrFasFAOperState OBJECT-TYPE
    SYNTAX      INTEGER {
      enabled(1),
        disabled(2)
        }
    MAX-ACCESS          read-only
    STATUS      current
    DESCRIPTION
        "This is the operational state of the feature. This has got several
        states.  For a feature to be enabled, it has to be activated,
        unlocked, and meet all constaints.  If a constraint is not met, such as
        Feature Key has exipired, the operational state shall be disabled.

        enabled (1):  indicates that the feature is enabled on the blade and
                in use by the application
        disabled (2): indicates that the feature is disabled on the blade.
                The feature is not in use in the application"
    ::= { usrFasFAEntry 7 }

usrFasFADiag    OBJECT-TYPE
    SYNTAX      INTEGER {
        noError (1),        -- Feature is enabled and in use
```

```
            termPending (2),      -- Feature is disabled but still in use
            initRequired (3),     -- Featue not in use, requires initialization for operation
            notSupported (4),     -- Feature is not supported (for downgraded s/w for example).
            locked (5),           -- Feature administratively prohibited
            verirfyFailed (6),    -- Feature verification failed, feature is prohibited
            wrongTime (7),        -- Feature out of bounds in regards to time
            activationFailed (8), -- Fature activation failed
            notActivated (9),     -- Feature is not Activated
            noUnits (10),         -- No more feature units available
            noFeature (11)        -- No feature key for feature
        }
    MAX-ACCESS          read-only
    STATUS      current
    DESCRIPTION
            "A diagnostic code."
    ::= { usrFasFAEntry 8 }


usrFasFARowStatus  OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS          read-write
    STATUS      current
    DESCRIPTION
        "Support for row creation is optional in FAS Sub-Agents.  Application
            shall  create a row for every supported feature.  RowStatus object is
            present in the MIB in case row creation is desired in the future.
            Objects in existing rows can be changed to activate and deactivate
            features without the rows themselves being created or deleted.
            inService and notInService values shall be used to control Feature Key
```

request and release.

active (1) - r/w

A Status of active means that the blade 'wants' permission for the feature represented by this row even if its some time in the future. At initialization time, the FAS Sub-Agent will communicate with the FAS Agent to make sure that it is permitted to use the feature in the configured time window. If the permission is denied, the row remains 'active'. At run time, setting this object to 'active' shall trigger the FAS Sub-Agent to initate a Feature Activation Request transaction to the FAS Agent. This transaction shall request/verify permission for the feature in the configured time frame even if the featureStatus was already 'active'. If application doesn't support the feature, then the transaction shall ensure that the permission is released from the card, and the status set to 'notInService'.

notInService (2) - r/w

Indicates that the blade does not 'want' permission for the feature represented by this row. The FAS Sub-Agent shall ensure that the permission is released from the card by performing a Feature Activation Request transaction with the FAS Agent. The transaction shall be performed even if the FAS Agent thinks it does not have permission for the feature.

notReady (3) - ro

Not be supported. This is a state stating the row is incomplete in dribble mode row creation. This state shall not happen because there are default values for every settable column.

createAndGo (4) - wo

Not supported.

createAndWait (5) - wo

Not supported.

destroy (6) -- wo

Not supported.

The set will fail for baselined features. Rows representing

baselined features can't be removed from this table.

"

DEFVAL { active }

::= { usrFasFAEntry 9 }

usrFasFAGroup OBJECT-GROUP

OBJECTS {

usrFasFADescription,

usrFasFAStartDate,

usrFasFAEndDate,

usrFasFAActState,

usrFasFAOperState,

usrFasFADiag,

usrFasFARowStatus

}

STATUS current

```
  DESCRIPTION
     "Feature Activation Table"
  ::= { usrFasSAGroups 1 }


usrFasFACompliance MODULE-COMPLIANCE
  STATUS       current
  DESCRIPTION
     "The compliance statement for FAS Sub-Agents."
  MODULE  -- this module
     MANDATORY-GROUPS { usrFasFAGroup }
  ::= { usrFasSACompliances 1 }


END -- end of module CW-FAS-SUBAGENT-MIB.
```

11 Detailed SYSLOG information

This section should include a detailed description of at least the following four levels of syslog criticality of events: CRITICAL, UNUSUAL, COMMON, and INFORMATIONAL.

11.1 CRITICAL level SYSLOG messages

fassa_nm.c:<line> () CfmInfo CfmStatus=<text status> 0x<hex status> parent=<oid> inst=<oid>

This indicates that a CFM load or store failed. The message indicates which part of the CFM load/store failed and why.

ARG CfmRequest CfmRequestType=<type>

Indicates whether the failing CFM was a load or store. 1 is CfmLoad, 2 is CfmSave

TCP Open to FAS Agent failed: <reason>

Indicates an internal failure occured while trying to establish a TCP connection from the FAS Subagent to the FAS Agent.

TCP Urgent received - Invalid in FAS Protocol

Indicates that a TCP packet marked "URGENT" was received. This is invalid in the FAS protocol. The FAS Subagent got possibly connected to the wrong application in the System Manager.

ARG Message type: <type in chr> (0x%<type in hex>)

Indicates the IPC that encountered a critical error during processing.

ERR <file>:<line> <function>() <errClass> <errId> <errText>

Indicates an internal error (fault) occured in the FAS Subagent.

PROP <file>:<line> <function>() <errClass> <errId> <errText>

Indicates the propagation of an internal error to the calling function. This gives a backtrace of the fault context.

<errClass> is one of the following:

ErrInvalidArgument

ErrInvalidMessage

ErrUnexpectedResult

ErrUnexpectedStatus

ErrViolatedState

ErrViolatedPrecondition

ErrViolatedInvariant

ErrViolatedPostcondition

<errId> can be one of the following:

Null Pointer

Invalid Pointer

Out of Bounds

"Backtrace..."

<errText> is some elaboration of the error.

11.2 VERBOSE level SYSLOG messages

FASSA fassa_main.c:<line> Message Received, Type : <chr IPC code> (0x<hex IPC code>).

Trace of IPC messages recevied by the FAS Subagent process

FASSA fassa_main.c:<line> Invalid Message Received, Type : <chr IPC code> (0x<hex IPC code>).

IPC messages recevied by the FAS Subagent process that it doesn't expect to handle.

FASSA Feature Activation[<featureID>].procedure_type <old> -> <new>

Setting the type of transaction [request or release].

FASSA Feature Activation[<featureID>].procedure_state <old> -> <new>

Trace of a FASP Transaction state change.

FASSA transact_state <old> -> <new>

Trace of a FasSa Transaction state change while connected to FAS Agent.

Connection to FAS Agent opened.

Indicates that a TCP Connection to the FAS Agent successfully reached the established state.

Connection to FAS Agent closed.

Indicates that a TCP Connection to the FAS Agent left the established state. It indicates that the connection started to close, it doesn't show wheter the close is complete yet.

TCP Connection to FAS Agent closed: <reason>.

Indicates that a TCP Connection finished closing with some status. The status is shown.

FASSA connection.tcp_state <old> -> <new>

Trace of TCP state change of connection from FAS Subagent to FAS Agent.

12 Debuging Facilities

This section should include a detailed description of the debugging facilities available to the developers and the testers other than the Syslog information, such as built-in test, MIB counters, and built-in traces, data structure audit, hidden commands. etc.

12.1 XXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

12.2 -D DEBUG flag

Compiling with the -D DEBUG flag will enable the tracing macros in the code that are placed in nearly all functions

FAS_TRACE_POINT ()

FAS_TRACE_ENTER ()

FAS_TRACE_EXIT ()

FAS_TRACE_INT (n)

FAS_TRACE_STR_int (str, n)

12.3 SET FACILITY "FAS Subagent" LOGLEVEL x

13 Unit Test Plan and Test Cases

Each module or feature that is developed in the Wireless projects will undergo unit testing. The unit testing to be carried out for this feature will be described in detail in this section. Please specify the unit test plan and test cases as complete as possible. This section will be reviewed with the peer engineers and ITG engieers.

Test Case #	Descrption	Input Specification	Output Specification
1	Able to automatically determine FAS Agent's IP Address	_SHOW FAS	System Manager's IP Address should be displayed as "FAS Agent ipaddr"
2	Show features that can be enabled on the TC2K HA and TC2K PDSN	LIST FEATURES SHOW ALL FEATURES _SHOW NMC FEATURE_MASK	All features that apply to the blade, and only the features that apply to the blade will be listed (no omissions, no additions)
3	Enable a feature (Positive test)	Make sure FAS Agent is able to grant a unit of feature 2 to that blade LIST FEATURES to verify blade doesn't have feature yet SET FACILITY "FAS Subagent" LOGLEVEL DEBUG ENABLE FEATURE 2	LIST FEATURES should have feature 2 enabled, Active, and Operational On System Manager, Verifty that feature was granted to blade Look at trace to verify packets and contents
4	Disable a feature (Positive test)	From step 3, DISABLE FEATURE 2	LIST FEATURES should show feature as Disabled, Inactive and Inop

			Look at trace to verify packets and contents
5	Enable a feature (Negative test)	Make sure FAS Agent is not able to grant a unit of feature 2 to that blade LIST FEATURES to verify blade doesn't have feature yet SET FACILITY "FAS Subagent" LOGLEVEL DEBUG ENABLE FEATURE 2	LIST FEATURES should show feature 2 as Disabled, Inactive and Inop Look at trace to verifty that request went to FAS Agent, and was denied by FAS Agent.
6	Verify persistance (Positive test)	Make sure FAS Agent is able to grant a unit of feature 2 to that blade LIST FEATURES to verify blade doesn't have feature yet ENABLE FEATURE 2 LIST FEATURES to verify blade has feature Active and Operational SAVE ALL equivalent on TC2K Reboot card Reboot chassis	LIST FEATURES after card reboot to verify that feature is till active LIST FEATURES after chassis reboot to verify that feature is still active

14 Memory And Performance requirement

Memory and perfromance requirements, if applicable, should be detailed in this section.

15 Other Considerations and Contraints

Any other considerations that are not covered in the previous sections should be included in this section.